VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Analýza textu pomocí neuronových sítí

Text Analysis Using Neural Networks

Jan Kusák

Bakalářská práce

Vedoucí práce: doc. Ing. Jan Platoš, Ph.D.

Ostrava, 2021

Abstrakt

Tato práce popisuje metody pro analýzu textů založené na hlubokém učení a neuronových sítí. Cílem této práce je popsat již existující metody pro analýzu textu a ukázat je v praxi s propojením neuronovými sítěmi. V práci se nejdříve zabývám metodami analýzy textu, poté neuronovými sítěmi a nakonec konkrétní implementací na praktické ukázce.

Klíčová slova

metody analýzy textu, neuronové sítě

Abstract

This thesis describes methods for text analysis based on deep learning and neural networks. The aim of this thesis is to describe the existing methods for text analysis and show them in practice with the interconnection of neural networks. In my thesis at first I describe methods of text analysis. Then neural networks and finally an implementation on a practical demonstration.

Keywords

methods of text analysis, neural networks

Poděkování

Rád bych poděkoval panu doc. Ing. Janu Platoš, Ph.D. za velmi vstřícný přístup, jeho trpělivost a spoustu užitečných rad, díky kterým jsem byl schopen svou práci vypracovat. Také své rodině za veškerou podporu a motivaci.

Obsah

Seznam použitých zkratek a symbolů

NN	–	Neuronová síť
RNN	–	Recurrent neural network
LSTM	–	Long Short-Term Memory
GRU	–	Gated Recurrent Unit
GloVe	–	Global Vectors for Word Representation
BERT	–	Bidirectional Encoder Representations from Transformers
CBOW	–	Continuous Bag-of-words
NLP	–	Natural language processing
tanh	–	Hyperbolický tangens

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Cílem této práce je popsat metody pro analýzu textu a následné odzkoušení těchto metod pomocí Neuronových sítí. Pro absolvování tohoto tématu jsem se rozhodl z důvodu, abych si vyzkoušel práci s umělou inteligencí, zjistil mnoho nových a zajímavých témat, technik, metod atd. Chtěl jsem alespoň částečně porozumět tématu umělé inteligence a druhů práce na kterých se umělá inteligence podílí. Ačkoli během mého studia na Vysoké škole Báňské, jsem si prošel velice zajímavými předměty a některými méně, tak mě trochu mrzela absence umělých inteligencí, což mě zajímalo už od střední školy. Proto tato práce byla pro mě požehnání a já jsem velice rád, že se mi naskytla. Tato práce pojednává o umělé inteligenci a její nasazení s rozborem a následného pochopení, nám lidem, normálnímu textu. Během této práce jsem se naučil neskutečně mnoho informací z okruhu umělé inteligence, ale také i s tématikou NLP. Ovšem jako každá práce potřebuje mít i ukázky a ty Vám poskytnu v kapitole praktické ukázky. Po dosažení této kapitoly, už by měli být jasné, jaké metody a architektury sítí používám a co dané metody a architektury dělají. Na závěr jsou zhodnoceny dosažené cíle a vše okolo toho.

Kapitola 2

Natural Language Processing - NLP

Natural Language Processing(NLP) [1] [2] neboli zpracování přírozeného jazyka je soubor různých technik jako je například lingvistika, informatika a umělá inteligence, která se zabývá interakcí mezi strojovým a lidským jazykem. Zjednodušeně jak naprogramovat počítač ke zpracování a analýze velkých dat přirozeného jazyka. Ve výsledku je počítač schopen porozumět kontextu v textu. Výzvy pro NLP se často týkají speech recognition(rozpoznávání řeči), natural language understanding(porozumění přirozeného jazyka) a natural language generation(generování přirozeného jazyka). NLP má bohatou historií a za tu dobu vzniklo mnoho technik na všelijaké věci. Ovšem v mé práci se projeví obvzlášť jedna technika s názvem Sentiment Analysis.

2.1 Sentiment Analysis

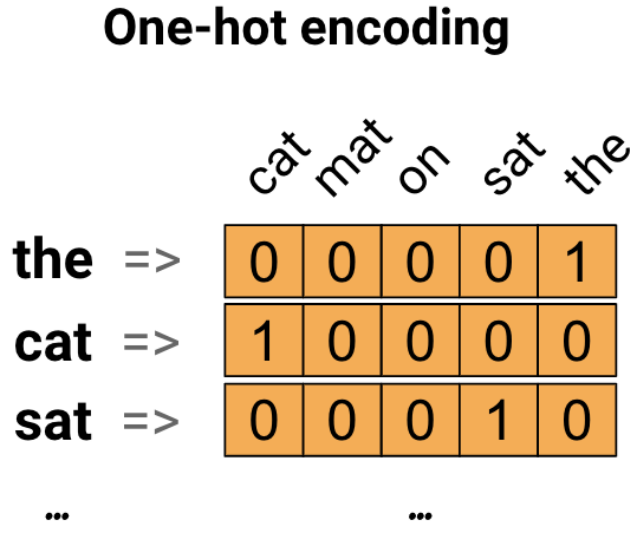
Sentiment Analysis [3] je jedna z nejvíce používaných technik v NLP. Sentiment analysis je užitečná v případě zpracování nějakých dotazníku, komentářů, nebo recenzí, u kterých tuto techniku použiji. Nejjednodušším výstupem Sentiment analysis je výstup o těchto možných hodnotách, pozitivní, negativní a neutrální. Pro provedení Sentiment Analysis potřebujeme převést text do něčeho co stroje porozumí. Tento proces může být komplikovaný.

2.2 Metody analýzy textu

Techniky jako jsou strojově učené algoritmy preferují jasně stanovené vstupní data, tak aby byly pevné délky. Tyto algoritmy nemohou přímo pracovat s textem, tak jak jej známe. Proto text musí být přeměněn do podoby, ve které si s ní tyto algoritmy poradí. Např. převod textu do číselné podoby, vektorů čísel. Těchto metod existuje celá řada.

2.3 One-hot encoding

Jako první metoda, kterou popíšu je metoda one-hot encoding. Tato metoda překládá slova pomocí tabulky, neboli slovníku, kdy za dané slovo se dosadí nulový vektor, s velikostí slovníku a na určeném místě(indexu) se nastaví hodnota 1, která reprezentuje dané slovo. Toto je jeden způsob jak se dají převést slova do číselné podoby. Vše má svůj háček a ani u této metody to není jiné. Představte si větu, která obsahuje tisíc slov, to by znamenalo, že pro každé slovo musíme vytvořit unikátní nulový vektor s hodnotou 1 na místě indexu pro dané, slovo a také, že co slovo, to vektor s jednou jedničkou a zbytkem nul. Proto tato metoda je neefektivní kvůli své vlastnosti, s každým přidaným slovem se musí velikost všech vektorů ve slovníku zvýšit a tím i přidat nulu, která nic nereprezentuje.



Obrázek 2.1: One-hot encoding [4]

Následně ukážu na ukázce, jak vypadá implementovaný příklad one-hot encoding

```
# Převod textu pomocí One-hot encoding
text_corpus = ["I was the best of times",
               "I was the worst of times",
               "I was the age of wisdom",
               "I was the age of foolishness"]

unique_words = set()
for seq in text_corpus:
  for word in seq.split():
    unique_words.add(word)
```

```
vocab = {}
for _, word in enumerate(unique_words):
  value = np.zeros(len(unique_words))
  value[_] = 1
  vocab[word] = value
```

Listing 2.1: One-hot encoding v praxi

V této ukázce máme hlavní text. Nejdříve se vytvoří slovník. Abychom slovník vytvořili potřebujeme znát unikátní slova, která uložíme do setu $unique - words$. Poté už jsme schopno vytvořit slovník, kde jako klíč nám poslouží právě zjištěná unikátní slova a jeho hodnota bude 1D vektor velikosti součtu unikátních slov, který bude vyplněn nuly a pouze na místě indexu, který je přiřazen slovu bude umístěna jednička. Následně je vyzobrazeno jak vypadá převod pro slovo 'best'.

```
Slovo: "best" je převedeno na: [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Obrázek 2.2: One-hot encoding výsledek

2.4 Přiřazení každému slovu unikátní číslo

Tato metoda už je zajímavější a hlavně efektivnější. Její princip se nijak extra neliší od metody one-hot encoding. Hlavní změna, a to proč je efektivnější, je vyměnění nulového vektoru s jedničkou na místě indexu, celočíselným číslem pomocí slovníku, tomu se také říká tokenizování. Tato metoda už je o něco lepší, každopádně i přesto tato prezentace slov pomocí unikátních slov nedokáže zachytit jakýkoli vztah mezi slovy. Na následujícím kódu je ukázána možná ukázka.

```
# Převod textu pomocí Unique number
text_corpus = ["I was the best of times",
               "I was the worst of times",
               "I was the age of wisdom",
               "I was the age of foolishness"]

vocab = {}
for seq in text_corpus:
  for word in seq.split():
    if word not in vocab:
      vocab[word] = len(vocab)

transformed = []
```

```
for seq in text_corpus:
  tokenized = []
  for word in seq.split():
    tokenized.append(vocab[word])
  transformed.append(tokenized)
```

Listing 2.2: Unique number encoding v praxi

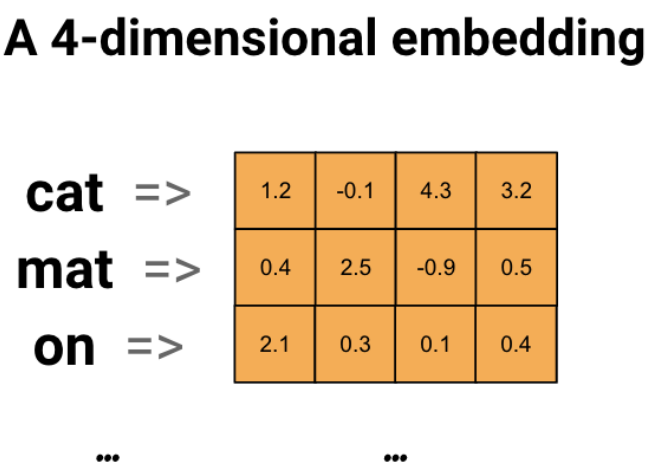
V této ukázce máme hlavní text. Nejdříve se vytvoří slovník, ve kterém se každému unikátnímu slovu připíše hodnota, která bude dané slovo reprezentovat. Následně je vyzobrazeno jak vypadá převod pro větu.

```
Věta: "I was the age of wisdom" je převedena na: [0, 1, 2, 7, 4, 8]
```

Obrázek 2.3: Unique number encoding výsledek

2.5 Word embeddings

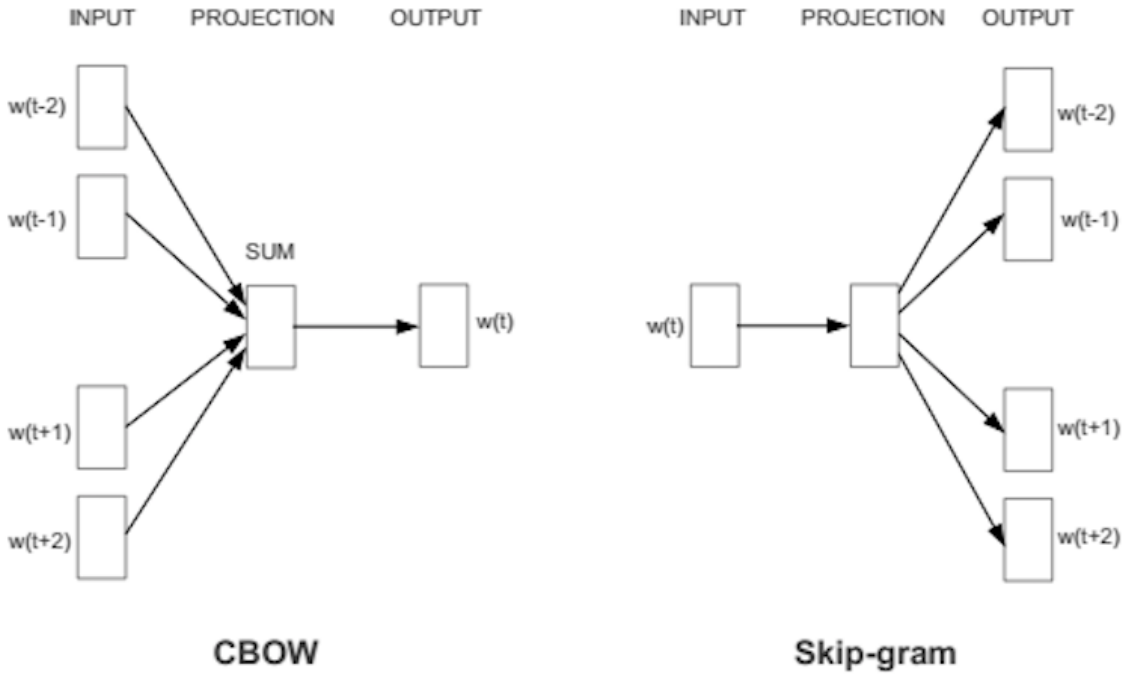
Word embeddings(vkládání slov) metoda funguje také pomocí slovníku, ale nepřekládá slova do nulového vektoru ani do celočíselného čísla. Naopak je překládá do vektoru desetinných čísel. Zde se může zdát nejasné, jak se vlastně k hodnotám desetinných čísel dojde? Nu, o to se nestaráme my manuálně, ale tento úkol přebírá model strojového učení, kdy vektory desetinných čísel využije jako váhy a podle nich, jak se bude model trénovat, se také budou i hodnoty v těchto vektorech měnit. Toto je neskutečná výhoda oproti předchozím metodám, kde se natvrdo zvolí vektor, nebo číslo pro dané slovo a to zůstane neměnné. Word embedding je vlastně high-dimensional reprezentace slov podle kontextu ve kterém se různá slova objevují. Pomocí Word embeddings jsme schopni porovnat podobnosti slov a poskytnout více užitečné informace NLP modelům.



Obrázek 2.4: Word embeddings [4]

2.6 Word2Vec

Jedna z metod používající Word embeddings je od Googlu vytvořený model Word2Vec [5]. Word2Vec je skupina nebo spíše rodina algoritmů, pomocí kterých je možné úspěšně převést text do strojové podoby i s určitými vztahy mezi slovy. Word2Vec je dvou vrstvá neuronová síť, která zpracovává text pomocí vektorizování slov. Kde na vstupu je hlavní text a výstup poté je sada vektorů, která reprezentuje daná slova. Díky tomu nám Word2Vec převede slova do číselné podoby, které neuronové sítě dokáží porozumět. U Word2Vec je možné si vybrat jeden ze dvou modelů pro učení se reprezentaci slov. Continuous Bag-Of-Words(CBOW) a Continuous Skip-gram.



Obrázek 2.5: Word2Vec představení možností [5]

2.6.1 Continuous Bag-Of-Words - CBOW

Continuous Bag-Of-Words [6] je model, který zkouší předpovědět slovo podle jeho okolí, což jsou slova, která se nachází před a za daným slovem. U této metody nehraje role pořadí slov proto se tato metoda jmenuje bag-of-words. Intuice je taková, sekvence slov jsou podobné pokud mají podobný obsah. Právě z obsahu můžeme pochopit význam sekvence. Na následujícím kódu ukážu jak se převádí text do vektorové podoby pomocí BOW [7].

```
# Převod textu pomocí BOW
text_corpus = ["I was the best of times",
               "I was the worst of times",
               "I was the age of wisdom",
               "I was the age of foolishness"]

vocab = set()
for seq in text_corpus:
  for word in seq.split():
    vocab.add(word)

vectorized = []
```

```
for seq in text_corpus:
  tokens = []
  for word in vocab:
    tokens.append(1 if word in seq else 0)
  vectorized.append(tokens)
```

Listing 2.3: BOW v praxi

V této ukázce máme hlavní text v podobě kterou dobře známe. Ovšem stroje ne a proto se musí převést na vektory. Nejdříve se vytvoří slovník, slovník zde není nic jiného než zapsaná univerzální slova do proměnné *vocab*. Jakmile je slovník načten a my víme o všech slovech v našem hlavním textu, převedeme text do vektorové podoby. Vektorová podoba vypadá následovně.

```
Věta: "I was the best of times" je převedena na: [1, 1, 1, 1, 0, 0, 1, 0, 0, 1]
```

Obrázek 2.6: BOW: Ukázka převodu

Jak můžeme vidět věta se nám převedla do vektoru. Tento vektor má velikost našeho slovníku. Hodnoty 1 a 0 se doplňují za slova, která jsou obsažena v dané větě. Je to velice podobný princip s přirovnáním ku one-hot encoding, s tím že se nedosazuje vektor za slovo ale za celou větu/sekvenci.

Bohužel má to své nevýhody. Růstem slovník roste i velikost vektoru reprezentující danou sekvenci. Obdobný problém vyskytující se u one-hot encoding. Proto se vyvíjí úsilí jak daný slovník zmenšit. Např. ignorování interpunkce, ignorování častých slov, která na sebe stejně neváží nějaký velký význam jako jsou slova $'a', 'of', 'the'$, oprava přepsaných slov, kdy se třeba uživatel omylem přepsal, zmenšování slov k jejich kořenům (slovo $'playing'$ se stane slovem $'play'$). Toto jsou docela jednoduché úpravy. Existuje i komplexnější úprava. Vytvoření slovníku, který bude obsahovat sloučená slova neboli n-gram. Tato metoda pomůže nejenom zredukovat velikost slovníku, ale také zachytí význam daných slov.

V této metodě každé slovo, token se nazývá *gram*, pokud budeme chtít vytvořit slovník, ve kterém se budou objevovat dvě spojená slova pak budeme pracovat s bigramy. Ačkoliv vytvoříme skupiny dvouslovné, tak to ještě neznamená, že v našem slovníku nebudou samostatné slova. Pouze se pokusí najít slova, která významově patří k sobě.

Pochopení *gramu*, *gram* se většinou vyskytuje v podobě n-gramu, kdy za n se dosadí libovolné číslo (pokud je hodnota čísla 1, pak máme unigram, pokud je hodnota čísla 2, pak máme bigram atd.). Na sekvenci slov: new york city, si ukážeme a lépe vysvětlíme n-gramy. Pokud bychom chtěli z takové sekvence získat unigramy pak by naše sekvence slov vypadala takto: [”*new*”, ”*york*”, ”*city*”]. Bigramy by přehodnotily unigramy a přetvořily by výsledek na [”*newyork*”, ”*city*”]. Zde můžeme vidět spojení dvou unigramů do jednoho bigramu. Samozřejmě můžeme pokračovat dále a to trigramem, který by výsledek změnil do podoby [”*newyorkcity*”]. Ovšem toto je pouze ukázka jak n-gramy fungují.

2.6.2 Skip-gram

Jako druhá metoda, co se dá použít pro vytvoření Word2Vec je metoda skip-gram [8]. Rozdíl mezi touto metodou a metodou CBOW je v jejich předpovědí. CBOW pomocí okolí předpovídá jaké slovo by se mohlo dosadit doprostřed mezi okolí, zatímco skip-gram metoda předpovídá jaké by mohlo být okolí daného slova.

Skip-gram reprezentuje kontext slova jako páry n-gramů. Za n-gram si můžeme představit samostatné slovo. Model Word2Vec, který je trénován právě pomocí skip-gram algoritmu používá jako trénovací data n-gramy.

Pro ukázku si skip-gram představíme na následující větě: The wide road shimmered in the hot sun. Tato věta obsahuje 8 slov a pro každé slovo budeme hledat skip-gramy pomocí definovaného okolí, definované okolí je hodnota jak daleko máme hledat okolní slova, co se slučují s určitým slovem. Toto definované okolí se nazývá window size. Na obrázku 2.7 můžeme vidět, jak určení skip-gramů vypadá

Window Size	Text	Skip-grams
2	[The **wide** road shimmered] in the hot sun.	wide, the wide, road wide, shimmered
	The [wide road **shimmered** in the] hot sun.	shimmered, wide shimmered, road shimmered, in shimmered, the
	The wide road shimmered in [the hot **sun**].	sun, the sun, hot

Obrázek 2.7: Skip-gram vybírání n-gramů [9]

Cílem trénování skip-gramů je co nejlépe vyjádřit okolí daných slov, což tak trochu připomíná vztah mezi slovy. Pro tento cíl se dá zapsat jako průměrná pravděpodobnost logaritmu. Pro pochopení následujícího textu, si prosím představte větu. Je zapotřebí se na větu podívat jako na sekvenci slov co jdou po sobě. Např. Dneska je krásný den. Toto je sekvence následujících se slov, která začínají prvním slovem $'Dneska'$ a následuje $'je'$ atd. Pro právě tyto sekvence $w1, w2, ...wt$ se využívá následující formule.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

Obrázek 2.8: Skip-gram formule [9]

- c znamená velikost kontextu

Základní skip-gram formulace definuje pravděpodobnost pomocí softmax funkce. O této funkci se v mé práci již nezmíním. Ovšem její účel je zvýraznit nejvyšší hodnoty

- Máme hodnoty: $[1, 2, 3, 4, 1, 2, 3]$

Po provedení softmax funkce se nám z hodnot stane

- Hodnoty po zpracování: $[0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175]$

Po sečtení těchto hodnot by nám mělo vyjít 1 což značí 100%.

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^{\top} v_{w_I}\right)}$$

Obrázek 2.9: Skip-gram softmax [9]

V této softmax funkci:

- za v se dosadí vektory určitých slov, podle kterých se zjišťuje jejich okolí/kontext
- za $v´$ se dosadí vektory kontextových slov
- Tyto vektory v a $v´$ reprezentují slova
- za w se dosazuje velikost slovníku, neboli počet unikátních slov, která se nám vyskytují v datech/datasetech co našemu Word2Vec poskytneme na trénování.

2.6.3 CBOW, Skip-gram porovnání

Když existuje více metod, je vždy dobré vědět jaké jsou její rozdíly v celkovém porovnání. Např. CBOW je rychlejší, jelikož zachází s celým kontextem jako s jednou entitou. Zatímco Skip-gram vytváří různé tréninkové páry pro každé kontextové slovo. Na druhou stranu Skip-gram poskytuje lepší výsledky a tím zlepšuje i pravděpodobnost našeho modelu.

2.7 GloVe - Global Vectors for Word Representation

Další model, který se dá využít pro semantickou analýzu je GloVe [10]. Mezi Word2Vec a GloVe jsou jisté rozdíly. Liší se tím, že Word2Vec je prediktivní model, zatímco GloVe je založen na počtu výskytů(count-based). Modely založené na počtu výskytu, učí své vektory pomocí redukce dimenzí na matici počtů společných výskytů. Nejdříve si v podstatě vytvoří velikou matici společně se vyskytujících informací. Tzn. pro každé slovo se počítá jak často se vyskytuje v nějakém kontextu v naších datech. Poté danou matici převede na matici nižší dimenze, kde každý řádek poskytuje vektorovou reprezentaci pro každé slovo. V případě GloVe, matice počtů je znormalizována podle počtu výskytu a následně je upravena pomocí log-smoothing technice. Každopádně s porovnáním mezi GloVe a Word2Vec si oba modely vedou docela podobně. Výhoda u GloVe je lehčí paralelizace implementace modelu. Což znamená lehčí trénování nad daty a to je velice dobré.

2.8 BERT - Bidirectional Encoder Representations from Transformers

Jeden z moderních přístupů je BERT [11]. Tento model je založen na feature-based trénování. Kdy je zapotřebí již přetrénovaný model co produkuje word embedding, který je pak používán právě jako features v NLP modelu. BERT model využívá Tranformeru(Transformer), attention mechanism (mechanismus pozornosti), který se učí kontextové vztahy mezi slovy a podslovy v textu.

V původní verzi, Transformer poskytuje dva oddělené mechanismy. Čtení textového vstupu pomocí kodéru a vytvoření předpovídání zajišťuje dekodér. Jelikož úkolem BERTu je vygenerovat jazykový model, jediný nutný mechanismus poskytnutý Transformerem je enkodér.

Vysvětlení Bidirectional Encoder. Na rozdíl od directional(směrových) modelů, které čtou vstupní text postupně zleva do prava a nebo naopak, Transformer enkodér čte celou sekvenci slov najednou. Proto se tomu říká Bidirectional, neboli z obou směrů.

2.9 Word2Vec, GloVe, BERT rozdíly

Všechny tyto modely plní své úkoly dobře, ale BERT je plní ze všech nejlépe [12] [13]. Exisuje více důvodů proč tomu tak je. Hlavní důvody jsou pořadí slov není zahozeno a způsob generování word embeddings.

2.9.1 Pořadí slov

Word2Vec a GloVe word embeddings jsou kontextově nezávislé. Tzn. výstup těchto modelů je pouze jeden 1D vektor (embedding) pro každé slovo, spojující všechny různé smysly daného slova vyskytující se v hlavním textu. BERT dokáže generovat různé word embeddings pro slovo, které zachycuje kontext slova, právě díky své pozici ve větě. Ve výsledku Word2Vec a GloVe nezachycují pozici slov ve větách, ale BERT ano.

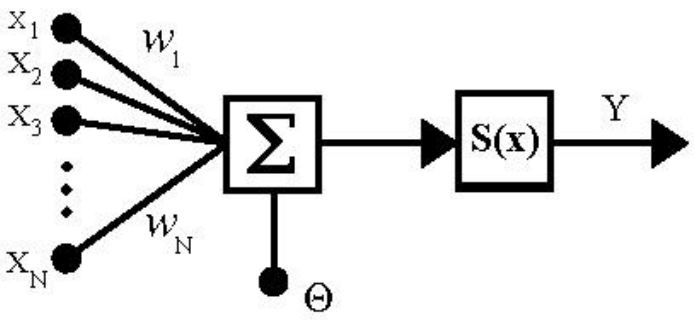
2.9.2 Generování embeddings

Word2Vec a GloVe jsou word-based modely, takže jako vstupní data dostávají slova a výjde z nich jako výstup word embedding. BERT na druhou stranu, reprezentuje vstup jako podslova a učí embeddings pro podslova. Ve výsledku BERT, který se učil nad datasetem obsahující milióny unikátních slov, obsahuje slovník s velikosti kolem třiceti tisíc. Neskutečně redukovaná velikost slovníku, ještě navíc pokud to porovnáme s Word2Vec a GloVe, kdy každé unikátní slovo má své místo ve slovníku. Reprezentování vstupu jako podslova a ne jako slova, je skvělý balance mezi slovní a znakovou reprezentací. Nejdůležitější benefit reprezentování vstupu jako podslova je vyhnutí se OOV(out of vocabulary) případů, kterými Word2Vec i GloVe trpí.

Kapitola 3

Neural Network - NN

Neural network [14] [15] neboli neuronová síť je jeden z výpočetních modelů používaných v umělé inteligenci(AI). Vzorem neuronové sítě je náš lidský mozek. Ovšem s porovnáním našeho mozku neuronová síť se skládá pouze z neuronů a cest(váh) mezi neurony. Neurony získávají vstupní data od jiných neuronů, tyto vstupní data poté zpracují pomocí aktivačních funkcí a následně je přepošlou dál dalším neuronům. Neuron má libovolný počet vstupů, ale pouze jeden výstup.



Obrázek 3.1: Model neuronu [14]

Na obrázku 3.1 lze vidět, jak vypadá model neuronu.

$$Y = S(\sum_{i=1}^{N}(w_i x_i) + \Theta)$$

Obrázek 3.2: Formule neuronu [14]

Toto je formule podle McCulloch–Pitts modelu neuronu.

- x: vstupní data z jiných neuronů

- w: váhy cest mezi neurony
- Θ: threshold obvykle pod jménem bias
- $S(x)$: aktivační funkce
- Y: výstup neuronu

Velikost váh vyjadřuje uložení zkušeností do neuronu. Čím je vyšší hodnota, tím je daný vstup důležitější.

3.1 Učení neuronové sítě

Cílem neuronové sítě je nastavit síť, tak aby dávala pokud možno co nejlepší výsledky. Toto trénování se uskutečňuje pomocí změny ve váhách. Trénování se rozlišuje na supervised, unsupervised.

3.1.1 Supervised

Neuronové síti je předložen vzor. Na jehož základě pomocí aktuálního nastavení je zjištěn výsledek, který je porovnán s požadovaným výsledkem a následně se určí chyba. Chyby se snažíme co nejvíce minimalizovat. Toto se provádí právě změnou váh a biasů. Toto opakujeme až do dosažení námi tolerované chyby, kdy můžeme danou neuronovou síť uznat za adaptovanou.

3.1.2 Unsupervised

Narozdíl od supervised učení, zde nevyhodnocujeme výstup. Jelikož nám výstup není znám. Neuronová síť v tomto případě dostává vstupní sady vzorů, které si sama třídí. Buď si vzory třídí do skupin nebo si přizpůsobí topologii vlastnostem vstupu.

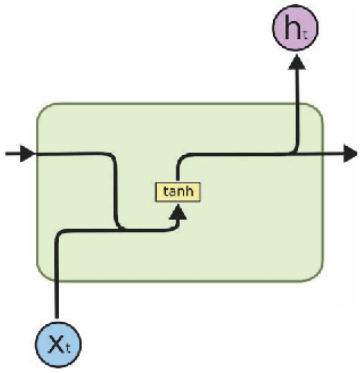
3.2 Recurrent Neural Netword - RNN

Existuje spousty architektur neuronových sítí a každá architektura se hodí na jiný úkol. Pro náš budeme potřebovat využít rekurentní neuronovou síť [16]. RNN jsou navrženy pro práci se sekvenčními daty. Sekvenčními daty se rozumí text, zvuk, video atd.

RNN používají předešlou informaci ze sekvence pro vytvoření výstupu. Můžeme si to představit na této větě: Já jsem člověk. Na počátku se vezme slovo 'Já' a vloží se do sítě, kde RNN vytvoří výstup. V dalším kroku se vezme slovo 'jsem' a hodnota z předchozího kroku a vloží se do sítě. Teď RNN má informaci o obou slovech 'Já' i 'jsem'. Dále tímto stejným procesem se zpracuje celá naše věta. Na samotném konci naše RNN má informace o veškerých předchozích hodnotách.

3.2.1 Architektura RNN

Architektura rekurentní neuronové sítě jednoho neuronu vypadá následovně.



Obrázek 3.3: RNN architektura [17]

Neuron RNN má podobné vlastnosti jako neuron v neuronové sítí. Do neuronu jdou vstupní data a data z předchozích kroků. Nad těmito daty se provede aktivační funkce tanh(Hyperbolický tangens)

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Obrázek 3.4: RNN výstup/hidden state [17]

RNN bohužel čelí jednomu problému a to s long-term dependencies, neboli s dlouhodobou závislosti. Toto je způsobeno vanishing gradient problémem, neboli mizejícím gradientem, pomocí kterého se mění váhy naší RNN.

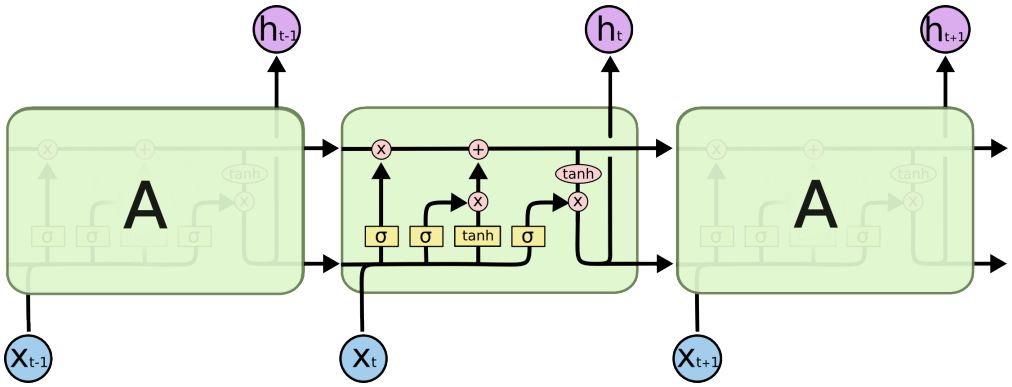
Ve své podstatě s mizejícím gradientem se přestává naše RNN trénovat a při každém dalším trénování zůstávají váhy stejné. Toto způsobuje již zmíněný problém s dlouhodobou závislosti, kdy RNN není schopna něčeho takového dosáhnout, ačkoli podle teorie by mohla. Hlavním problémem je složitost pro RNN uchovat si informaci po mnoho krocích.

Abychom předešli tomuto problému, byly vytvořeny specializované verze RNN. LSTM(Long Short-Term Memory) a GRU (Gated Recurrent Unit). Tyto rekurentní neuronové sítě používají konceptu gates(brán). Brány jsou používány pro kontrolu probíhajících informací v síti. Brány jsou schopné se naučit, které vstupy v sekvencí jsou důležité a uchovávají je v paměti.

3.3 Long Short-Term Memory - LSTM

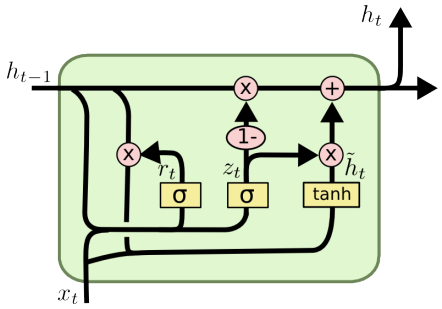
Long Short-Term Memory sítě jsou speciální verzí rekurentní neuronové sítě. Byly představeny Hochreiter & Schmidhuber v roce 1997. LSTM je specificky navrženo pro předejití problému s

dlouhodobou závislosti. Pamatovat si informaci po velice dlouhou dobu je ve své podstatě jejich základní chování.



Obrázek 3.5: LSTM architektura [18]

Architektura oproti RNN je o dost komplexnější. Důvodem jsou již zmíněné brány. LSTM obsahuje konkrétně tři brány.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Obrázek 3.6: LSTM architektura neuronu [18]

LSTM v sobě ukrývá Input, Output a Forget brány. Tyto brány používají jako aktivační funkci sigmoid, tudíž veškeré hodnoty těchto brán jsou v rozmezí 0 a 1.

3.3.1 Forget Gate

Tato brána kontroluje co zůstane uchováno a co naopak bude zapomenuto z minulých stavů. Jednoduše se rozhoduje jak moc informací z předchozího stavu by mělo být zachováno.

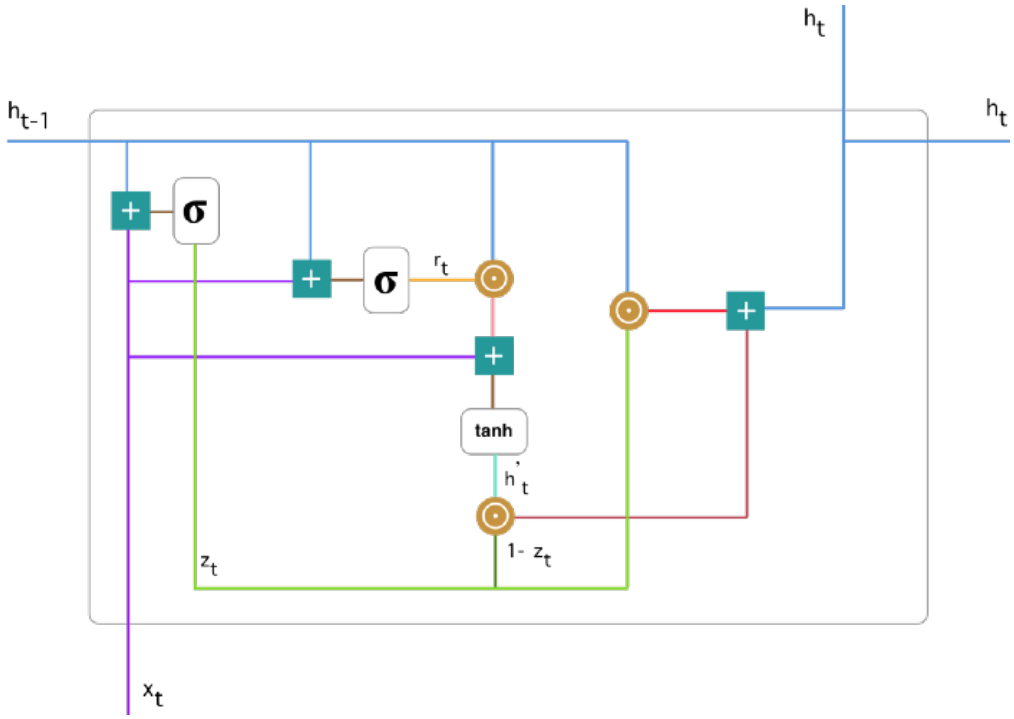
3.3.2 Output Gate

Tato brána kontroluje, které části neuronu jsou výstupem do skrytého stavu. Rozhoduje co bude příštím skrytým stavem.

3.4 Gated Recurrent Unit - GRU

GRU stejně jako LSTM je speciální verzí rekurentní neuronové sítě. GRU byla představena Cho, et al. v roce 2014. GRU se zaměřuje na vyřešení vanishing gradient problem, neboli problém mizejícího gradientu, což je hlavní problém RNN. GRU může být považováno za variaci LSTM, jelikož obě verze jsou podobně navržené a v nějakých případech produkují stejné výsledky.

Na vyřešení problému mizejícího gradientu standardních RNN, GRU používá Update Gate a Reset Gate. Ve své podstatě tyto brány reprezentují dva vektory, které rozhodují co za informace by měli projít do výstupu. Tyto brány mohou být trénovány, aby udrželi informaci z dávných kroků, aniž by se časem pozměnily, nebo úplně smazaly.



Obrázek 3.7: GRU architektura neuronu [19]

Obrázek 3.8: GRU význam operací [19]

Opět můžeme vidět komplexní architekturu s porovnáním u rekurentní neuronové sítě. Stejně jako LSTM můžeme v této architektuře zpozorovat brány. U GRU jsou zde brány konkrétně dvě. Update Gate a Reset Gate.

3.4.1 Update Gate

Update Gate pomáhá modelu se rozhodnout jak moc minulé informace je zapotřebí přesunout do budoucnosti. Díky tomuto se model může rozhodnout zkopírovat veškeré informace z minulých kroků a zároveň je eliminován risk vzniknutí vanishing gradient problému.

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

Obrázek 3.9: GRU Update Gate [19]

3.4.2 Reset Gate

Reset Gate v podstatě je použitá k rozhodnutí kolik informací z minulosti(minulých kroků) bude zapomenuto.

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

Obrázek 3.10: GRU Reset Gate [19]

3.4.3 Porovnání

LSTM a GRU vznikly kvůli vanishing gradient problému u RNN. Otázka ale je, jak si tyto architektury povedou se sentiment analýzou nad datasetem v IMDB a s klasifikací nad datasetem Reuters. Tyto porovnání jsou vyzobrazeny a popsány v praktické ukázce.

Kapitola 4

Praktické ukázky

Uvažoval jsem nad čím bych mohl reprezentovat analýzu textu. Nakonec po radě vedoucího jsem vytvořil tyto praktické ukázky. Semantickou analýzu recenzí nad datasetem IMDB a klasifikaci článků nad datasetem Reuters. Tyto ukázky ukazují rozdíly mezi různými metodami analýzy textu a i mezi architekturami sítě.

Samostatné ukázky lze nalézt v příloze, kde je k celému kódu vytvořen komentář pro každou věc, co a proč v ukázkách dělám. I tak zde popíši o co v praktických ukázkách jde a jak jsem postupoval.

Veškeré mé ukázky jsem vyvíjel v prostředí Colab Research od Googlu [20]. Pokud si budete mé ukázky zkusit rozjet, doporučuji to právě v tomto prostředí. Stačí jednoduše nahrát kód a spustit. Nemusíte se trápit s instalací balíčků a dalšími nastaveními.

4.1 Klasifikace článků nad datasetem Reuters

V této praktické ukázce, jsem si vyzkoušel vytvořit klasifikaci článků nad datasetem Reuters. Mým cílem tedy bylo.

- Vytvořit si modely s architekturou GRU a vyzkoušet si na nich analýzu textu pomocí Word2Vec s metodami Skip-gram a CBOW. Pro porovnání s klasickými metodami bylo zapotřebí vytvořit model, který bude používat metodu one-hot encoding.
- Nad nejlepší metodou analýzy textu vytvořit další dva modely s architekturami LSTM a RNN. Posléze nad těmito architekturami provést porovnání, která se nejlépe vyrovná s Loss(chybou), která bude mít nejlepší přesnost výsledků nad testovacími daty a která z nich je vlastně nejrychlejší.

Postupoval jsem postupně podle těchto bodů.

- Nejdříve jsem si stáhnul dataset (Reuters) z keras datasetu a přerozdělil data na dvě potřebné části (trénovací a testovací)

- Poté jsem si načetl slovník daného datasetu, jelikož dataset už přichází zpracovaný, kdy je rozdělen do sekvencí a ztokenizován budu muset tuto operaci vrátit, abych měl správné data pro trénování mého Word2Vec modelu

- Převedl jsem si tokeny ze sekvencí na slova, která představují a následně jsem si vyzkoušel metodu bigramů [21]

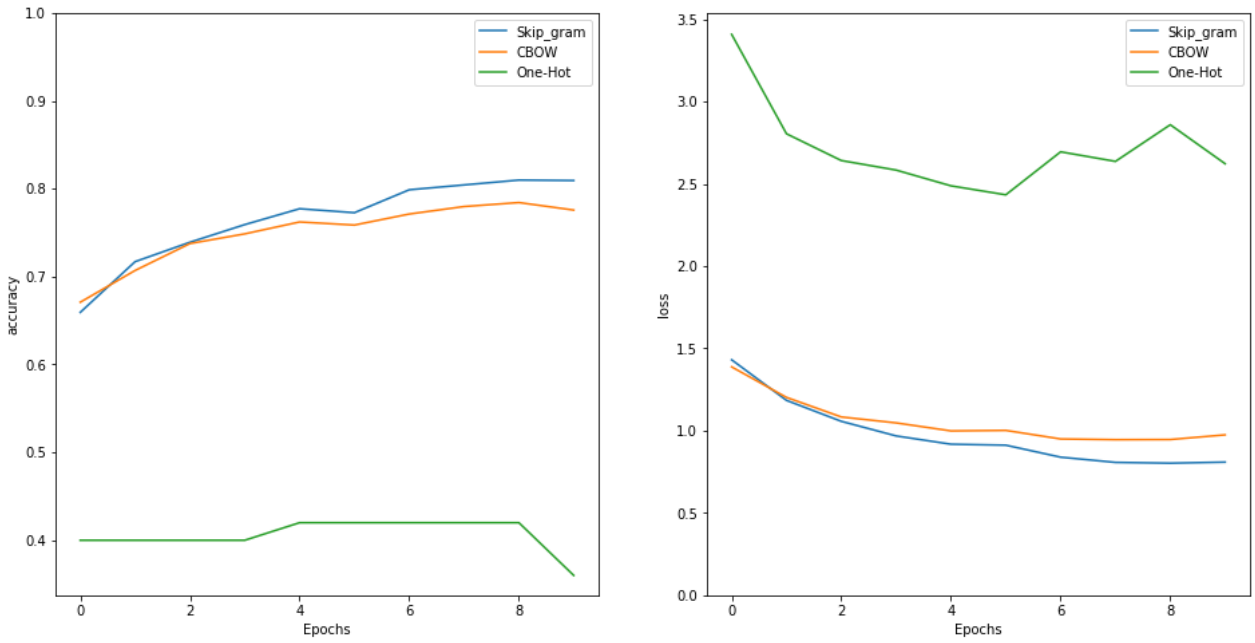
- Následovalo vytvoření a trénování Word2Vec modelu pro skip-gramy tak i pro CBOW nad biagramy sekvencí

- Pro použití Word2Vec bylo zapotřebí vytvořit další slovník, pro Embedding vrstvu (id/token: 1D vektor určité velikosti/reprezentace slova)

- Jakmile metoda pro převod textu pomocí skip-gramů a CBOW byla vytvořená. Potřeboval jsem ještě metodu One-hot encoding. Pro kterou bylo zapotřebí vytvořit unikátní slovník pro převod textu do podoby One-hot encoding.

- V této chvíli jsem měl vše už předpřipravené a mohl jsem se vrhnout na modely. Vytvořil jsem si modely pro klasifikaci Reuters datasetu, těchto modelů dohromady je pět. Kde první tři z nich jsou modely architektury bidirectional(obousměrné) GRU, s využitím metod pro analýzu textu(Skip-gram, CBOW, One-hot encoding). Čtvrtý a pátý model jsou architektur bidirectional RNN a LSTM s použitím metody Skip-gram, která se ukázala být tou nejlepší.

Díky tolika dostupných modelů jsem byl chopen výsledky ukázat ve vizuální podobě, podle které se krásně porovnává. Konkrétně jsem porovnal rozdíl mezi samostatnými metodami analýzy textu a mezi architekturami modelu.

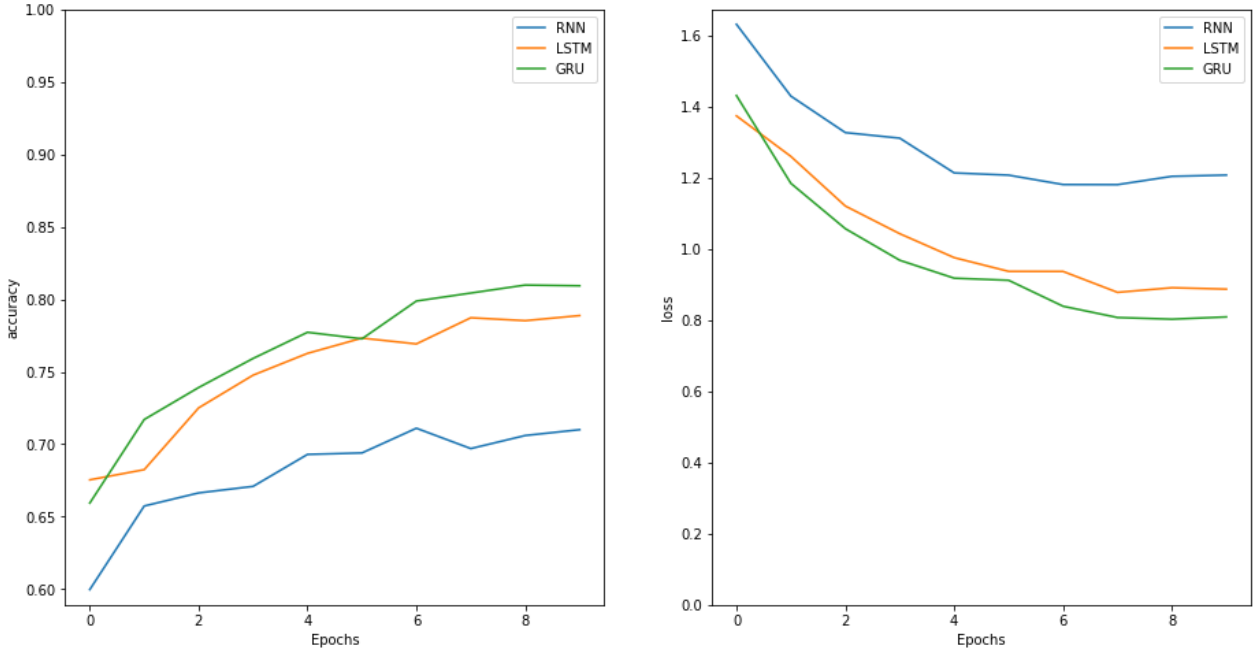


Obrázek 4.1: Porovnání metod analýzy textu

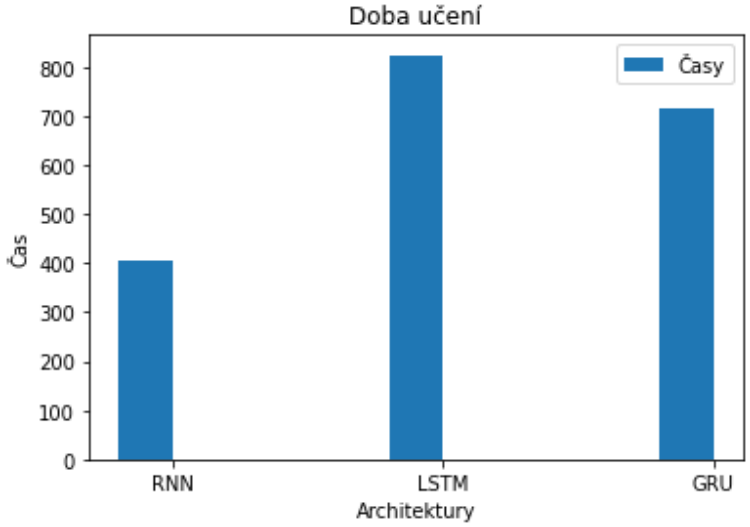
Teď k samostatným výsledkům. Jak lze vidět, na prvním obrázku 4.1 s vyhodnocenou přesností, porovnáním metod analýzy textu, tak v klasifikaci, si nejlépe vedl model, který používal metodu skip-gram. Na druhém místě je CBOW a na posledním je One-hot encoding.

S jedním problémem. One-hot encoding vyžaduje tolik paměti, že jsem nebyl schopen veškerý dataset projet modelem One-hot encoding a proto dataset, který používal One-hot encoding měl pouze velikost padesáti sekvencí, jak pro trénování tak pro testování.

Na druhém obrázku pořadí naších modelů zůstalo stejné. Jelikož modely se snaží co nejvíce minimalizovat Loss(chyby), je tedy 'nejlepším' modelem opět se skip-gramem po kterém následuje CBOW a nakonec One-hot encoding.



Obrázek 4.2: Porovnání architektur sítí



Obrázek 4.3: Časové porovnání v trénování

Tady 4.2 už jsou výsledky samostatných architektur. Na prvních dvou obrázcích můžeme vidět, že LSTM a GRU jsou určitě skvělé verze RNN a obě verze skutečně řeší problém RNN. Ale za jakou cenu? Pokud se podíváme na obrázek s časy 4.3, můžeme vidět RNN jako nejrychleji vytrénovaný model, kde GRU je na druhé pozici a LSTM na poslední. Tady záleží na nás co chceme více, rychlost nebo přesnost?

4.2 Sentiment analysis recenzí nad datasetem IMDB

V této praktické ukázce, jsem si vyzkoušel vytvořit sentiment analysis recenzí nad datasetem IMDB. Mým cílem tedy bylo.

- Vytvořit si modely s architekturou GRU a vyzkoušet si na nich analýzu textu pomocí Word2Vec s metodami Skip-gram a CBOW. One-hot encoding v tomto případě nepoužiji, právě kvůli tomu jak moc zabírá místo. Není dobré porovnávat modely, které byly trénovány nad různými datasety a proto One-hot encoding vynechávám.

- Podobně jako u klasifikace. Nad nejlepší metodou analýzy textu vytvořit další dva modely s architekturami LSTM a RNN. Posléze nad těmito architekturami provést porovnání, která se nejlépe vyrovná s Loss(chybami), která architektura bude mít nejlepší přesnost výsledků nad testovacími daty a která z nich je vlastně nejrychlejší.

Postupoval jsem postupně podle těchto bodů.

- Nejdříve jsem si stáhnul dataset (IMDB) z oficiálního zdroje, který jsem nalezl v dokumentaci tensorflowu [22]. datasetu a přerozdělil data na dvě potřebné části (trénovací a testovací)

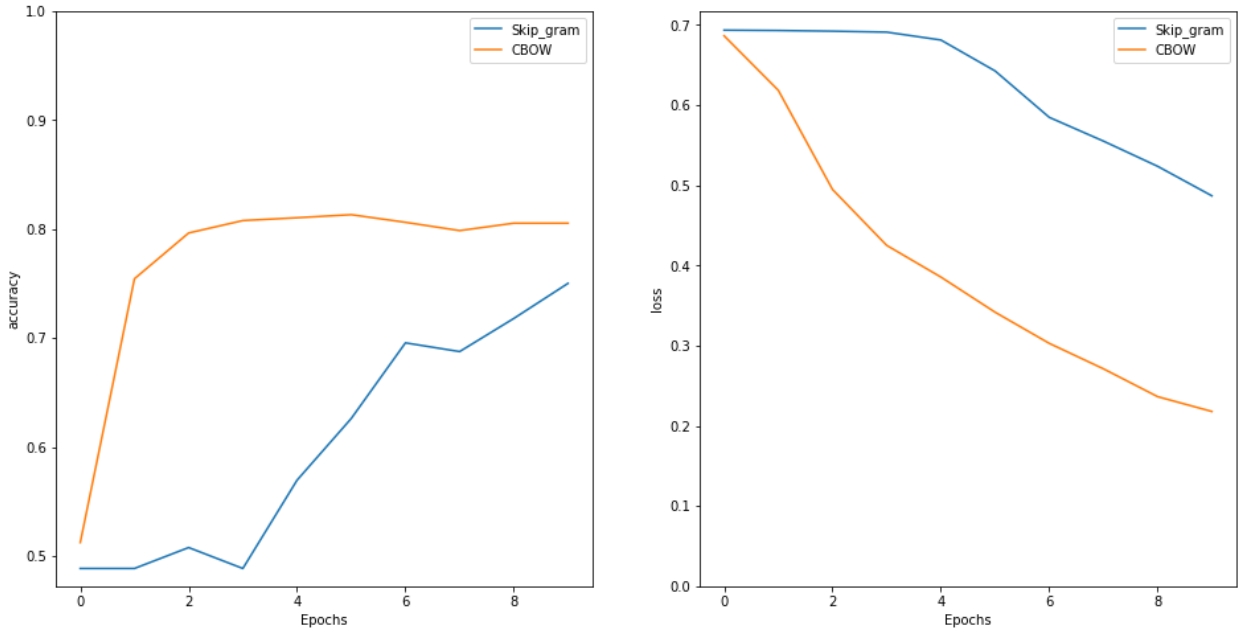
- Zpracoval jsem dataset a převedl věty na sekvence slov

- Následovalo vytvoření a trénování Word2Vec modelu pro skip-gramy tak i pro CBOW nad sekvencemi slov

- Pro použití Word2Vec bylo zapotřebí vytvořit další slovník, pro Embedding vrstvu (id/token: 1D vektor určité velikosti/reprezentace slova)

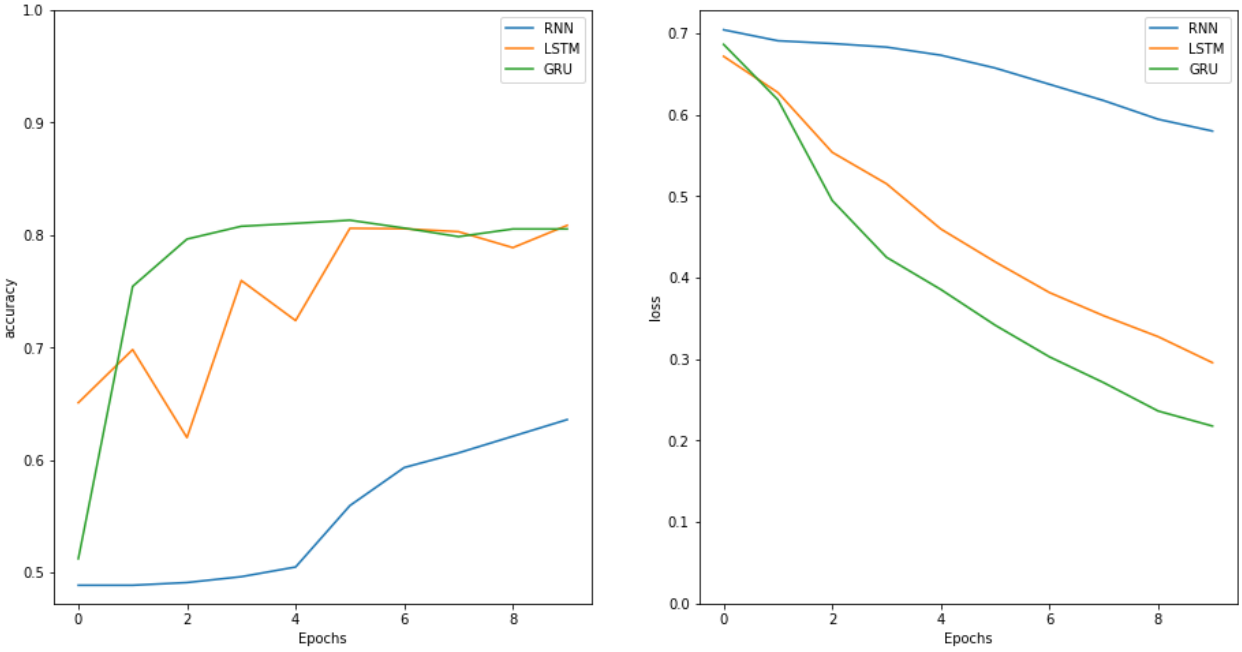
- V této chvíli jsem měl vše už předpřipravené a mohl jsem se vrhnout na modely. Vytvořil jsem si modely pro sentiment analysis IMDB datasetu, tyto modely jsou dohromady čtyři. Kde první dva z nich jsou modely architektury bidirectional(obousměrné) GRU, s využitím metod pro analýzu textu(Skip-gram, CBOW). Třetí a čtvrtý model jsou architektur bidirectional RNN a LSTM s použitím metody CBOW. CBOW pro změnu, abych nepoužil stejnou metodu pro klasifikaci tak i pro sentiment analysis.

Díky tolika dostupných modelů jsem byl chopen výsledky ukázat ve vizuální podobě, podle které se krásně porovnává. Konkrétně jsem porovnal rozdíl mezi samostatnými metodami analýzy textu a mezi architekturami modelu.

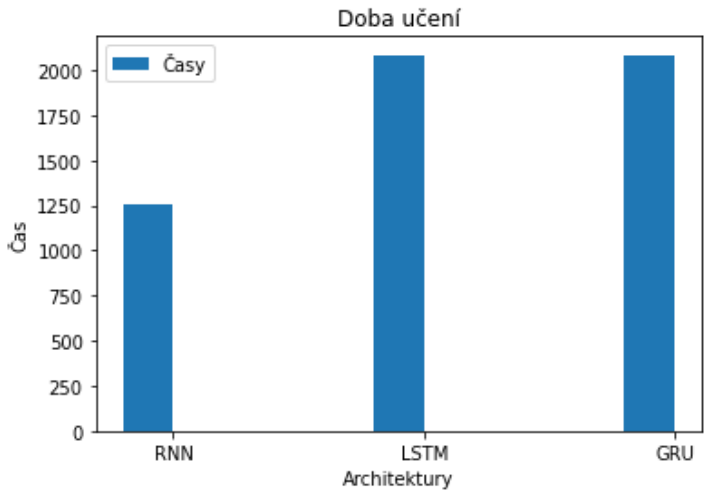


Obrázek 4.4: Porovnání metod analýzy textu

Kupodivu, nejlepší výsledky se dostavily modelu, který používal metodu CBOW. Je dost možné, že by Skip-gram výsledek otočil pokud bych mu dopřál více epoch na cvičení. Každopádně, podle obrázku 4.4 jako nejlepší ze dvou metod analýzy textu je CBOW a následně Skip-gram.



Obrázek 4.5: Porovnání architektur sítí



Obrázek 4.6: Časové porovnání v trénování

Na obrázku 4.5 lze vidět jak si vedli samostatné architektury s metodou CBOW. Ačkoli už nás nepřekvapí, že nejlépe si vedla architektura GRU, která měla nejmenší Loss a následovala architektura LSTM a RNN. Tak je zajímavé se podívat na časy daných architektur 4.6. Lze jde opět vidět velikou rychlost RNN architektury, každopádně s porovnáním jak si vedly architektury GRU a LSTM v klasifikaci nad datasetem Reuters a zde sentiment analysis nad datasetem IMDB je veliký rozdíl u architektury GRU, která se v tomto případě cvičila skoro stejnou dobu jako LSTM. To samozřejmě nezvěstí nic špatného, jenom je to zajímavost na kterou jsem chtěl poukázat.

A nakonec jsem slíbil vyzkoušení vyhodnocení nad naší vlastnoručně napsanou recenzi. Toto vyhodnocení se provádí nad architekturou GRU s CBOW jako metodou analýzy textu. Vyhodnocení jsem provedl nad těmito recenzemi.

- První věta: The movie was awful since start. The animation and the graphics were terrible. I would not recommend this movie at all.
- Druhá věta: The movie was cool. The animation and the graphics were out of this world. I would recommend this movie.

Tabulka 4.1: Výsledky vyhodnocení

Věta	Vyhodnocení	Textově
První věta	-8.19	negativni recenze
Druhá věta	3.58	pozitivni recenze

V tabulce 4.1 můžeme vidět, jak dopadlo naše vyhodnocení vět.

Kapitola 5

Závěr

5.1 Shrnutí

V této práci jsem vytvořil několik rekurentních neuronových modelů různých architektur. Použil jsem architekturu RNN, která byla ze všech nejrychlejší, LSTM, která ukázala své výhody a také GRU, které byla skvělá jak v rychlosti, tak i výsledky. Všechny tyto architektury byly odzkoušeny na různých způsobech analýz textu. Ať už Word2Vec s použitím Skip-gramů nebo CBOW. Samozřejmě jsem si vyzkoušel i vymodelovat síť pomocí one-hot encoding metody, to ale moc nedopadlo.

Nakonec jsem tyto modely vyzkoušel na dvou různých datasetech pro dvě různé techniky. Modely zkoušené nad datasetem Reuters vytvářeli klasifikaci článků, zatímco modely zkoušené nad datasetem IMDB vytvářeli sentiment analysis recenzí.

5.2 Možná vylepšení

Toto téma je složité. Oblasti kolem neuronových sítí a NLP, jsou tak obrovské, že jsem se sotva zakousnul. Ani nevím, co vše ještě existuje a co bych mohl použít pro vylepšení mých modelů. Určitě bych mohl využít metodu analýzy BERT, která by měla o něco lépe trénovat vazby mezi slovy, s tím bych začal a odtuď bych se odrážel dál.

5.3 Zkušenost

Ponořit se do neuronových sítí a nevědět jak hluboko se ponořím. Byla nakonec skvělá zkušenost. Určitě budu dále se zabývat tímto tématem. Podle mého, umělá inteligence je naše budoucnost a já jsem velice rád, že jsem se do tohoto témata dostal.

Literatura

1. *Natural language processing* [online] [cit. 2021-04-25]. Dostupné z: https://en.wikipedia.org/wiki/Natural_language_processing.
2. *Zpracování přirozeného jazyka* [online] [cit. 2021-04-25]. Dostupné z: https://cs.wikipedia.org/wiki/Zpracov%C3%A1n%C3%AD_p%C5%99irozen%C3%A9ho_jazyka.
3. VAIDYA, Neeraja. *5 Natural Language Processing Techniques for Extracting Information* [online] [cit. 2021-04-25]. Dostupné z: https://blog.aureusanalytics.com/blog/5-natural-language-processing-techniques-for-extracting-information.
4. *Word embeddings* [online] [cit. 2021-04-25]. Dostupné z: https://www.tensorflow.org/tutorials/text/word_embeddings.
5. *A Beginner's Guide to Word2Vec and Neural Word Embeddings* [online] [cit. 2021-04-25]. Dostupné z: https://wiki.pathmind.com/word2vec.
6. PHD, Jason Brownlee. *A Gentle Introduction to the Bag-of-Words Model* [online] [cit. 2021-04-25]. Dostupné z: https://machinelearningmastery.com/gentle-introduction-bag-words-model/.
7. DUBEY, Praveen. *An introduction to Bag of Words and how to code it in Python for NLP* [online] [cit. 2021-04-25]. Dostupné z: https://www.freecodecamp.org/news/an-introduction-to-bag-of-words-and-how-to-code-it-in-python-for-nlp-282e87a9da04/.
8. DOSHI, Sanket. *Skip-Gram: NLP context words prediction algorithm* [online] [cit. 2021-04-25]. Dostupné z: https://towardsdatascience.com/skip-gram-nlp-context-words-prediction-algorithm-5bbf34f84e0c.
9. *Word2Vec* [online] [cit. 2021-04-25]. Dostupné z: https://www.tensorflow.org/tutorials/text/word2vec.
10. JEFFREY PENNINGTON Richard Socher, Christopher D. Manning. *GloVe: Global Vectors for Word Representation* [online] [cit. 2021-04-25]. Dostupné z: https://nlp.stanford.edu/projects/glove/.

11. HOREV, Rani. *BERT Explained: State of the art language model for NLP* [online] [cit. 2021-04-25]. Dostupné z: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

12. RAJASEKHARAN, Ajit. *What are the main differences between the word embeddings of ELMo, BERT, Word2vec, and GloVe?* [Online] [cit. 2021-04-25]. Dostupné z: https://www.quora.com/What-are-the-main-differences-between-the-word-embeddings-of-ELMo-BERT-Word2vec-and-GloVe.

13. GOUWS, Stephan. *How is GloVe different from word2vec?* [Online] [cit. 2021-04-25]. Dostupné z: https://deeplearning.lipingyang.org/wp-content/uploads/2017/12/How-is-GloVe-different-from-word2vec_-Quora.pdf.

14. *Umělá neuronová síť* [online] [cit. 2021-04-25]. Dostupné z: https://cs.wikipedia.org/wiki/Um%C4%9Bl%C3%A1_neuronov%C3%A1_s%C3%AD%C5%A54.

15. HARRISON KINSLEY, Daniel Kukieła. *Neural Networks from Scratch in Python.* 2020.

16. KOSTADINOV, Simeon. *How Recurrent Neural Networks work* [online] [cit. 2021-04-25]. Dostupné z: https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7.

17. PEDAMALLU, Hemanth. *RNN vs GRU vs LSTM* [online] [cit. 2021-04-25]. Dostupné z: https://medium.com/analytics-vidhya/rnn-vs-gru-vs-lstm-863b0b7b1573.

18. *Understanding LSTM Networks* [online] [cit. 2021-04-25]. Dostupné z: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

19. KOSTADINOV, Simeon. *Understanding GRU Networks* [online] [cit. 2021-04-25]. Dostupné z: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be.

20. *Welcome To Colaboratory* [online] [cit. 2021-04-29]. Dostupné z: https://colab.research.google.com/.

21. HAMISH. *Training and plotting word2vec with bigrams* [online] [cit. 2021-04-25]. Dostupné z: https://www.kaggle.com/hamishdickson/training-and-plotting-word2vec-with-bigrams.

22. *An end-to-end open source machine learning platform* [online] [cit. 2021-04-25]. Dostupné z: https://www.tensorflow.org.